2.2 MOVEMENT

Movement is a combination of change in location and change in orientation. SWEG allows a user to cause simulated movement of a platform by assigning a sequence of positions to it. From a modeling perspective, there are two major aspects of movement: creation of a movement path subject to movement constraints, and the timing of the actual physical changes in location and/or orientation due to movement. Movement may also cause changes in the electromagnetic and acoustic signature of the mover. These effects are discussed in Section 2.3.

Movement Constraints

In the real world, movement is subject to laws of physics that define relationships between mass and motion; e.g., momentum, inertia, gravity, friction, etc. In addition to these dynamic limitations, real-world movement may be limited by propulsion capabilities, environmental conditions, operational rules, and component damage or failure.

Almost all limitations on movement in SWEG are defined by the user. Platforms in SWEG do not have mass, thus no dynamic limitations are imposed by the model. For example, the user can define the speed of a platform arbitrarily; SWEG does not check to see if it makes physical sense. Platforms stop instantaneously whenever the user instructions cause movement to stop; SWEG does not require coasting to the stop. After stopping, platforms may be removed or remain in the scenario. Even if a platform is airborne, it may stop in place and remain in the scenario for later actions, if the user so chooses. Fuel is considered only if the user chooses to do so and specifies the original amount of fuel and the rate of fuel usage.

SWEG has no internal assumptions about fuel usage or maximum speed, altitude, climb/dive rates, etc. It is the prerogative and the responsibility of the user to set the following limits as desired for each type of moving platform: fuel-usage, maximum acceleration, maximum deceleration, minimum turn radius, and climb/dive limits; the user may also override some of these limits in specific movement paths. Operational constraints may include terrain following, terrain avoidance, or threat avoidance; SWEG allows the user to select combinations of these options for each individual mover. The user may also define additional vertical boundary limits on the specific path, changes in orientation along the path, climb/dive rate and angle, and several other conditions. See the design requirements for a complete list of user options.

The basic movement path is defined by a start time and a list of successive positions and velocities for the mover. SWEG will create paths consisting of a sequence of straight line segments and arcs of circles that pass through the given path points, if possible given other user constraints. The path altitudes may be defined as above-ground-level (AGL) or mean-sea-level (MSL), the path mode as surface or 3D, and the path may be defined as a sequence of straight line segments with instantaneous turns or with turns defined by arcs of a specified minimum radius.

Movement may be either planned or reactive. SWEG allows the user to define preplanned movement path for platforms (except disaggregated platforms) and to define reactive deviations from that plan. The user may define reactive maneuvers for a disaggregated platform. The decisions used to cause reactive movement are discussed in Sections 2.9

through 2.11, the timing of position updates is the same in both modes, and limits on movement are nearly the same. See design requirement c for a complete list of options.

Real world platforms can start and stop and restart movement. This is also possible in SWEG. Platforms can start movement at any time after the start of the scenario simulation; however, they must exist (and thus be susceptible to detection and attack) from the start of the simulation unless they are disaggregated from another player at some time in the simulation. Platform paths can include preplanned stops and restarts, as well as reactive stops and starts. The user can choose to have platforms disappear after stopping or to remain in the scenario at the last point of their movement paths. Only if they remain after stopping can they restart movement.

Movement of one platform may be coordinated with movement of other platforms. Platforms such as aircraft, tanks, and ships may be moving in formation. SWEG allows the user to define formation movement of platforms if the platforms have implicit instantaneous communication with each other.

Timing of Movement

Movement may be continuous in the real world, but it must occur in discrete steps in a digital simulation model. A user defines a movement path by providing a list of successive positions and the speeds to be used in moving from one position to the next. The model must use this information to calculate the position of the platform at any specific point in time. Some models are time-stepped; i.e., new platform positions (and other changes in the scenario) are calculated at predetermined time intervals. Other models are event-stepped; i.e., positions (and other conditions) are calculated only at times they are needed for some internal or external purpose. SWEG is more like an event-stepped model, although the user must define a (global) minimum time interval between position updates.

SWEG does calculate position only when it is needed for some internal or external purpose, but the position calculated need not be the position at the current game time, it could be a position for some time in the future (positions at past times are calculated only for output). Internally, SWEG needs platform position to test for a possible interaction with another platform; preliminary tests are based on path and scenario geometry, additional tests are based on capabilities and decision-making rules. External requests for position can be generated by user instructions to provide this information for graphics or listing purposes. They can also be generated by other assets (simulators connected to SWEG) when SWEG is run in virtual mode.

Because of the way position updates are calculated and used within SWEG, the timing of movement is not applicable for this FE; only the creation of the movement path will be included as part of this section.

2.2.1 Functional Element Design Requirements

The design requirements necessary to implement the movement functional element in SWEG are listed below.

a. SWEG will provide a capability for a user to define movement constraints and characteristics for each type of platform in the TDB. The user options will include the following input data items.

1. Fuel usage defined as a burn rate dependent on altitude and speed.
2. Maximum acceleration.
3. Maximum deceleration.
4. Maximum traversable slope for a surface mover. (Not currently implemented)
5. Minimum turn radius defined as a 3-dimensional distance.
6. Maximum and minimum altitudes (MSL). (Not currently used)
7. Climb/dive limits defined as maximum and minimum vertical rates.
8. Orientation limits defined as the maximum roll, pitch, and yaw angles relative to the velocity vector. (Not currently used)
9. Orientation rates defined as maximum angular rates of change in each of roll, pitch, and yaw. (Not currently used)
10. Maximum and minimum speed limits (Not currently used)
11. Delay time between the decision to start moving and the actual start of physical movement

SWEG will impose these limits when creating a movement path for platforms of the given type in the SDB. SWEG will allow the user to override the acceleration and turn radius limits by specific SDB instructions, but will change the SDB positions if they violate the defined climb/dive limits for that type.

b. SWEG will provide a capability for a user to define a planned movement path for each non-disaggregated platform in the SDB. The path will include a time to start movement and sequence of locations. The locations may be defined in terms of Cartesian coordinates or latitude, longitude, and altitude. Path altitudes (or z-coordinates) must be defined as AGL or MSL. The path must be defined as surface or three-dimensional movement, and the turns must be specified as instantaneous with sharp corners or as requiring a minimum turn radius. The following additional options will be available; all except the first may be specified for each path point.

1. Minimum altitude AGL or maximum altitude MSL within a (2-dimensional) polygonal area
2. Orientation and rate of change of orientation in terms of roll, pitch, and yaw (if orientation is not specified, the platform orientation is defined according to the velocity vector)
3. Climb/dive rate or climb/dive angle
4. Terrain-following, terrain-avoidance, or threat-avoidance
5. Turn mode as an instantaneous turn (sharp corner) or as a turn in an arc that starts at the given path point
6. Speed and speed changes
7. Stop at this point for a specified time interval
8. Turn direction as right or left or the shorter of the two directions.

9. Turn radius in terms of distance or force (g's) or roll, pitch, and yaw (Only the distance option is currently used)
10. Turn type as banked or skid (Only the banked option is currently used)

SWEG will use these definitions to create a movement path consisting of straight line segments and arcs of circles, subject to constraints in the other design requirements.

c. SWEG will provide a capability for a user to define reactive movement paths for each type of platform in the TDB. The path will include a sequence of locations defined as relative Cartesian coordinates. The x and y coordinates may be relative to the mover or relative to the target; the z-coordinate may be defined as AGL, MSL, relative to the mover, or relative to the target. The following additional options for each path point will be available.

1. Orientation and rate of change of orientation in terms of roll, pitch, and yaw (if orientation is not specified, the platform orientation is defined according to the velocity vector)
2. Climb/dive rate or climb/dive angle
3. Terrain-following or terrain-avoidance or threat-avoidance
4. Turn mode as an instantaneous turn (sharp corner) or as a turn in an arc that starts at the given path point
5. Speed and speed changes.
6. Stop at this point for a specified time interval
7. Turn direction as right or left or the shorter of the two directions.
8. Turn radius in terms of distance or force (g's) or roll, pitch, and yaw (Only the distance option is currently used)
9. Turn type as banked or skid (Only the banked option is currently used)

SWEG will use these definitions to create a reactive movement path consisting of straight line segments and arcs of circles, subject to constraints in the other design requirements.

d. SWEG will provide a capability for a user to define formation movement for platforms that have implicit instantaneous communication with each other; i.e., for multiple platforms of a single player. The user will be able to define a formation template containing the following information for each echelon for each type of formation for each type of player.

1. Relative position in terms of Cartesian coordinates relative to the formation positions to be defined in the SDB.
2. Identification of the leader for this echelon; this may be another echelon or it may be the local origin of the formation.
3. Relative time to start a turn as either the same time as the leader starts the turn or no sooner than this echelon comes abreast of the leader's turn point, or the turn time specified for the formation in the SDB.

4. Echelon identification to assume after a turn; this can be the same echelon identification it had before the turn, or it can change its relative position within the formation and assume a different echelon identification.
5. Distance allowed after a turn to get back into its formation position.

For each instance of a formation defined in the SDB, a user will be able to define a planned movement path. The path will include a time to start movement and sequence of locations. The locations may be defined in terms of Cartesian coordinates or latitude, longitude, and altitude. Path altitudes (or z-coordinates) must be defined as AGL or MSL. The path must be defined as surface or three-dimensional movement, and the turns must be specified as instantaneous with sharp corners or as requiring a minimum turn radius. The following additional options will be available for each path point.

1. Speed and speed changes.
2. Turn radius in terms of distance or force (g's) or roll, pitch, and yaw (Only the distance option is currently used).

SWEG will use these definitions to create movement paths consisting of straight line segments and arcs of circles, subject to constraints in the other design requirements.

e. SWEG will use the path created to determine the position of a platform whenever the position is needed for some internal or external purpose or to fulfill user instructions for the maximum time between position updates. SWEG will calculate platform position no more frequently than is necessary to satisfy the preceding requirements.

2.2.2 Functional Element Design Approach

This section is not currently available.

2.2.3 Functional Element Software Design

This section contains a table and two software code trees which describe the software design necessary to implement the requirements and design approach outlined above. Table 2.2-1 lists most of the functions found in the code trees, and a description of each function is provided. Figure 2.2-1 describes the path to ORNJcontrol, the top-level C++ function in the code for movement. (The call to ORNJcontrol that creates the planned movement paths is not shown. That call occurs during the SDB step as part of parsing the user instructions, and only runtime functions are included here.) Figure 2.2-2 is the code tree for ORNJcontrol and its subordinate functions.

A function's subtree is provided within the figure only the first time that the function is called. The tree for DLG8control is completely expanded in Section 2.11 Logic Processes. Not all functions shown in the figures are included in the table. The omitted entries are trivial lookup functions (single assignment statements), list-processing or memory allocation functions, or C++ class functions for construction, etc.

TABLE 2.2-1. Movement Functions Table.

Function	Description
AKSNcontrol	controls resource allocation decisions
AKSNexecute	performs actions to carry out decisions
AKSNmaneuver	performs actions for reactive maneuvers
AKSNmnvrpfrm	reactively causes one of a player’s own platforms to move
AKSNrequests	performs actions for filling requests from other players
BaseHost::Run	runs all steps
conlink	connects contour line endpoints
conpoly	construct polygons for terrain/threat avoidance
crslwc	determines line/circle crossing
crslwl	determines line/line crossing
DLG8control	controls resource evaluation and selection decisions
geocrs	checks for line crossing with a set of lines
geoctr	determines 2-D center of gravity for a set of points
geodst	determine closest or farthest point to a given point
KNMXcontrol	adds new point to movement path
KNMXmaneuver	calculates distances to level off and start/stop acceleration
KNMXpoints	determines start/stop acceleration and level off points
KNMXturnarc	constructs an arc for a turn
lifbeg	begins a new player’s life
lifdna	creates a new player using a dummy player as a template
lifnew	creates a new life during model execution
main	controls overall execution
MainInit	initiates processing and runs either the boot step or normal execution
MainParse	controls parsing of user instructions
MovePlane::CalcRate	gets rate of turn
MovePlane::CalcValue	calculates a given value for orientation
MovePlane::GetDuration	gets duration time for a turn
MovePlane::GetValue	looks up a value for orientation
MovePlane::SetRate	sets the rate for changes in orientation
MovePlane::SetValue	sets the yaw, pitch, or roll value for orientation
movset	controls specific actions for reactive maneuvers
movstop	performs stop movement action
ORNJbestpath	finds best path using shortest distance as criterion
ORNJckpoints	processes path checkpoints for terrain following
ORNJcombpoly	combines two polygons including any internal holes

TABLE 2.2-1. Movement Functions Table. (Contd.)

Function	Description
ORNJcontrol	controls construction of the checkpoints of a movement path
ORNJcopypoly	makes a duplicate copy of an input polygon
ORNJeveryleg	sets up movement path for all legs defined by checkpoints
ORNJfollowpath	generates path, possibly using terrain following
ORNJgoaround	finds any paths around polygon obstruction
ORNJlinepoly	determines if a line segment intersects one polygon
ORNJmasking	generates masking pattern for use in threat avoidance
ORNJmazepath	finds shortest distance path through restricted areas
ORNJmergesmooth	merges path segments and smooths out unnecessary points
ORNJpolygons	combines any polygons that intersect
ORNJsmoothpath	smoothes path possibly creating new options
ORNJsubsets	determines if input polygons are a subset of each other
ORNJvertical	generates vertical portion of terrain following path
program	controls execution of all steps except bootstrap
region	determines if a point is within a two dimensional region
semant	controls semantic processing of instructions
simnxt	controls event sequencing and runtime execution
simphy	controls processing of physical events
simthk	controls processing of mental events
simul8	controls semantic processing of runtime instructions
srhpro	searches table for interval containing a specific value
TMaster::GetUanVocab	retrieves a user application name counter
TMBRmutigon	recycles multiple polygon tree structures
TMBRscratchpad	recycles scratch pad tree structures
TMemory::Allocate	allocates permanent storage
TMemory::AllocTemp	allocates temporary storage
TMemory::Deallocate	deallocates a list of blocks by using the address within the provided pointer
TMemory::LLSTremove	returns a pointer to the block on the traversed list which matches the provided key
TMemory::LLSTsearch	searches a list
TOrientation::AddManeuver	adds an orientation maneuver
TOrientation::AddPoint	adds a point to the orientation change list
TOrientation::CalcRollRate	calculates the roll rate
TOrientation::CalcYaw	calculates the yaw
TOrientation::FindElement	searches for an entry in the orientation list
TOrientation::FindOrCreate	adds or finds an entry on orientation list
TOrientation::GetLeftVector	returns the left facing vector

TABLE 2.2-1. Movement Functions Table. (Contd.)

Function	**Description**
TOrientation::NeedInitialFacing	determines if initial facing calculations are needed
TOrientation::RateChange	sets the orientation rate change
TOrientation::SetFacing	sets the facing for orientation
TOrientElement::DeleteAllAfter	deletes all orientation points after a time
TPathEntry::CheckOBOrient	checks orientation in the path under construction
TPathEntry::CreateOrientation	creates orientation points
TPathEntry::ManeuverData	looks up data associated with maneuver
TPlayer::Transform	transforms this player for model execution
TPlayer::Translate	translates data to set up this player
TTable::SearchInt	searches a table for a specific integer
TTerrain::AdjustBounds	adjusts the terrain bounds
TTerrain::BeenVisited	returns a flag if terrain vertex has been visited
TTerrain::EdgeMasklos	determines the masking of terrain edges
TTerrain::Elevation	determines the z-coordinate on a surface given an x, y coordinate pair
TTerrain::FindTriangle	determine the terrain triangle for a point given and x,y coordinate pair
TTerrain::FollowContour	follows a contour through the terrain
TTerrain::InBounds	checks for point within terrain bounds
TTerrain::IntermediatePoints	determines intermediate points for triangles
TTerrain::LineOfSight	determine if there is a line of sight between two objects
TTerrain::MarkAsVisited	marks the vertex as having been visited
TTerrain::Obstacles	determines obstacles for movement
TTerrain::RemoveMark	removes the visited mark for a terrain vertex
vecarc	calculates subtended angle for an arc defined by several vectors
wpnfyr	processes weapon intercept events
xl8fpl	generates RDB future path list block

```
main
    |-BaseHost::Run
        |-MainInit
            |-program
                |-MainParse
                    |-semant
                        |-simul8
                            |-simnxt
                                |-simthk
                                |   |-DLG8control
                                |       |-AKSNcontrol
                                |           |-AKSNexecute
                                |               |-AKSNmaneuver
                                |               |   |-AKSNmnvrpfrm
                                |               |       |-ORNJcontrol
                                |               |       |-movset
                                |               |           |-movstop
                                |               |               |-ORNJcontrol
                                |               |-AKSNrequests
                                |                   |-ORNJcontrol
                                |                   |-lifbeg
                                |                       |-lifnew
                                |                           |-lifdna
                                |                               |-TPlayer::Transform
                                |                                   |-TPlayer::Translate
                                |                                       |-xl8fpl
                                |                                           |-ORNJcontrol
                                |-simphy
                                    |-wpnfyr
                                        |-lifbeg
                                            |-lifnew
                                                |-lifdna
                                                    |-TPlayer::Transform
                                                        |-TPlayer::Translate
                                                            |-xl8fpl
                                                                |-ORNJcontrol
```

FIGURE 2.2-1. Movement Top-Level Code Tree.

```
ORNJcontrol
        |-TMemory::Index2Ptr
        |-TTable::SearchInt
        |-TMemory::Allocate
        |-srhpro
        |-TMaster::TerrainOn
        |-TMaster::GetTerrain
        |-TTerrain::Obstacles
        |     |-TTerrain::FindTriangle
        |     |-TTerrain::InBounds
        |     |-VertexIndex::VertexIndex
        |     |-VertexIndex::Value
        |     |-VertexIndex::operator++
        |     |-VerticeArray::operator[]
        |     |-TTerrain::BeenVisited
        |     |     |-SwegExcpt::SwegExcpt
        |     |     |-TTerrain::toIndex
        |     |     |     \-SwegExcpt::SwegExcpt
        |     |     \-VertexIndex::Value
        |     |-operator+
        |     |-TTerrain::FollowContour
        |     |     |-SwegExcpt::SwegExcpt
        |     |     |-TTerrain::BeenVisited
        |     |     |-VertexIndex::operator==
        |     |     |-isZeroEquiv
        |     |     |-TMemory::Index2Ptr
        |     |     |-TMemory::AllocTemp
        |     |     |-TMemory::Ptr2Index
        |     |     |-TTerrain::MarkAsVisited
        |     |     |     |-SwegExcpt::SwegExcpt
        |     |     |     |-TTerrain::toIndex
        |     |     |     |-VertexIndex::Value
        |     |     |     \-operator+
        |     |     |-VerticeArray::operator[]
        |     |     |-operator+
        |     |     |-TTerrain::InBounds
        |     |     |-TTerrain::AdjustBounds
        |     |     |     \-SwegExcpt::SwegExcpt
        |     |     |-VertexIndex::operator++
        |     |     \-VertexIndex::operator+=
        |     |-conpoly
        |     |     |-TMemory::Index2Ptr
        |     |     |-TMemory::Allocate
        |     |     |-TMemory::Ptr2Index
        |     |     \-TMemory::Deallocate
        |     |-TMemory::Index2Ptr
        |     |-TMemory::Deallocate
```

FIGURE 2.2-2. Movement Code Tree.

```
|   |   |-TMemory::Deallocate
|   |   \-TMemory::Ptr2Index
|   |-TTerrain::RemoveMark
|   |   |-SwegExcpt::SwegExcpt
|   |   |-TTerrain::toIndex
|   |   |-VertexIndex::Value
|   |   \-operator+
|   |-conlink
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::AllocTemp
|   |   |-TMemory::Ptr2Index
|   |   \-conpoly
|   \-TMemory::Ptr2Index
|-ORNJeveryleg
|   |-TMemory::Index2Ptr
|   |-DVector::DVector
|   |-operator-
|   |-DVector::GetHorizLength
|   |-TMemory::Ptr2Index
|   |-region
|   |   |-DVector::Getx
|   |   |-DVector::Gety
|   |   |-dbg_atan2
|   |   \-dist
|   |-TMemory::LLSTremove
|   |-TMemory::Deallocate
|   |-DVector::DVector
|   |-ORNJmasking
|   |   |-TMaster::GetTerrain
|   |   |-TMemory::Allocate
|   |   |-DVector::DVector
|   |   |-operator+
|   |   |-operator*
|   |   |-TMaster::TerrainOn
|   |   |-DVector::Putz
|   |   |-TTerrain::Elevation
|   |   |-TTerrain::LineOfSight
|   |   |   |-DVector::Getz
|   |   |   |-DVector::Getx
|   |   |   |-DVector::Gety
|   |   |   |-dist
|   |   |   |-dbg_acos
|   |   |   |-VertexIndex::VertexIndex
|   |   |   |-TTerrain::FindTriangle
|   |   |   \-TTerrain::EdgeMasklos
|   |   |       |-dist
|   |   |       |-VerticeArray::operator[]
|   |   |       |-operator+
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|   |   |       |-VertexIndex::operator+=
|   |   |       |-isZeroEquiv
|   |   |       \-TTerrain::toIndex
|   |   |-DVector::operator-=
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::Ptr2Index
|   |   |-DVector::Getx
|   |   |-DVector::Gety
|   |   \-geoctr
|   |-ORNJpolygons
|   |   |-ORNJcopypoly
|   |   |   |-TMemory::Allocate
|   |   |   |-TMemory::Index2Ptr
|   |   |   \-TMemory::Ptr2Index
|   |   |-ORNJsubsets
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-dbg_sqrt
|   |   |   |-dist
|   |   |   |-DVector::DVector
|   |   |   |-region
|   |   |   |-TMemory::Ptr2Index
|   |   |   |-ORNJlinepoly
|   |   |   |   |-DVector::DVector
|   |   |   |   |-DVector::DVector
|   |   |   |   |-dbg_sqrt
|   |   |   |   |-crslwc
|   |   |   |   |   |-operator-
|   |   |   |   |   |-DVector::Putz
|   |   |   |   |   |-operator^
|   |   |   |   |   |-dbg_sqrt
|   |   |   |   |   |-operator+
|   |   |   |   |   \-operator*
|   |   |   |   |-operator-
|   |   |   |   |-DVector::GetHorizLength
|   |   |   |   |-TMemory::Index2Ptr
|   |   |   |   |-crslwl
|   |   |   |   |   |-operator-
|   |   |   |   |   |-operator*
|   |   |   |   |   |-DVector::Getz
|   |   |   |   |   |-operator+
|   |   |   |   |   \-operator*
|   |   |   |   |-TMemory::Allocate
|   |   |   |   |-DVector::Getx
|   |   |   |   |-DVector::Gety
|   |   |   |   \-TMemory::Ptr2Index
|   |   |   |-DVector::Getx
|   |   |   |-DVector::Gety
|   |   |   \-DVector::operator=
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|   |   |-ORNJcombpoly
|   |   |   |-TMemory::Allocate
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-TMaster::DebugOn
|   |   |   |-TMemory::Ptr2Index
|   |   |   |-geoctr
|   |   |   |-geodst
|   |   |   |-TMemory::LLSTsearch
|   |   |   |-DVector::DVector
|   |   |   |-region
|   |   |   |-dist
|   |   |   |-ORNJlinepoly
|   |   |   |-DVector::operator
|   |   |   \-TMemory::Deallocate
|   |   |-TMemory::Ptr2Index
|   |   |-TMemory::LLSTsearch
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::Deallocate
|   |   \-TMemory::Deallocate
|   |-TMBRmutigon
|   |   \-TMemory::Deallocate
|   |-TMemory::LLSTsearch
|   |-ORNJmazepath
|   |   |-DVector::DVector
|   |   |-region
|   |   |-TMemory::Allocate
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::Ptr2Index
|   |   |-TMemory::LLSTsearch
|   |   |-ORNJlinepoly
|   |   |-TMemory::Deallocate
|   |   |-ORNJgoaround
|   |   |   |-TMemory::Ptr2Index
|   |   |   |-TMemory::LLSTsearch
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-dist
|   |   |   |-DVector::DVector
|   |   |   |-region
|   |   |   |-TMemory::Allocate
|   |   |   |-TMemory::Deallocate
|   |   |   \-ORNJsmoothpath
|   |   |       |-DVector::DVector
|   |   |       |-DVector::DVector
|   |   |       |-ORNJmergesmooth
|   |   |       |   |-DVector::DVector
|   |   |       |   |-dist
|   |   |       |   |-TMemory::Index2Ptr
|   |   |       |   |-ORNJlinepoly
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|   |   |       |   |-TMemory::Ptr2Index
|   |   |       |   |-DVector::DVector
|   |   |       |   |-region
|   |   |       |   |-operator+
|   |   |       |   \-operator-
|   |   |       |-crslwl
|   |   |       |-DVector::operator=
|   |   |       |-geocrs
|   |   |       |   |-DVector::DVector
|   |   |       |   |-DVector::DVector
|   |   |       |   |-DVector::operator
|   |   |       |   \-crslwl
|   |   |       |-TMemory::Deallocate
|   |   |       |-TMemory::Ptr2Index
|   |   |       |-TMemory::LLSTsearch
|   |   |       \-ORNJlinepoly
|   |   |-TMemory::Deallocate
|   |   \-ORNJbestpath
|   |       |-TMemory::Index2Ptr
|   |       |-dist
|   |       |-TMemory::Allocate
|   |       |-TMemory::Ptr2Index
|   |       |-DVector::DVector
|   |       |-DVector::Getz
|   |       |-FloatVector::operator=
|   |       |-DVector::DVector
|   |       \-TMBRscratchpad
|   |           \-TMemory::Deallocate
|   |-ORNJfollowpath
|   |   |-TMaster::GetTerrain
|   |   |-TMemory::Index2Ptr
|   |   |-isZeroEquiv
|   |   |-TPathEntry::TPathEntry
|   |   |-TPathEntry::SetTDepart
|   |   |-TPathEntry::SetTArrive
|   |   |-TPathEntry::SetPos
|   |   |-DVector::DVector
|   |   |-TPathEntry::PutUnitVel
|   |   |-operator-
|   |   |-TPathEntry::GetPos
|   |   |-DVector::Norm
|   |   |-TPathEntry::SetSpeed
|   |   |-TMemory::Ptr2Index
|   |   |-TMaster::TerrainOn
|   |   |-FloatVector::Putz
|   |   |-TTerrain::Elevation
|   |   |-DVector::GetHorizLength
|   |   |-ORNJckpoints
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|    |    |    |-TMemory::Deallocate
|    |    |    |-ORNJvertical
|    |    |    |    |-TMaster::GetTerrain
|    |    |    |    |-TTerrain::IntermediatePoints
|    |    |    |    |    |-DVector::DVector
|    |    |    |    |    |-DVector::Getx
|    |    |    |    |    |-DVector::Gety
|    |    |    |    |    |-dist
|    |    |    |    |    |-VertexIndex::VertexIndex
|    |    |    |    |    |-TTerrain::FindTriangle
|    |    |    |    |    |-VerticeArray::operator[]
|    |    |    |    |    |-operator+
|    |    |    |    |    |-VertexIndex::operator+=
|    |    |    |    |    |-TMemory::Index2Ptr
|    |    |    |    |    |-isZeroEquiv
|    |    |    |    |    |-TMemory::Allocate
|    |    |    |    |    |-FloatVector::Putx
|    |    |    |    |    |-FloatVector::Puty
|    |    |    |    |    |-FloatVector::Putz
|    |    |    |    |    |-TMemory::Ptr2Index
|    |    |    |    |    \-FloatVector::Getz
|    |    |    |    |-TMemory::Deallocate
|    |    |    |    |-TMemory::Ptr2Index
|    |    |    |    \-TMemory::LLSTsearch
|    |    |    |-dist
|    |    |    \-dbg_sqrt
|    |    |-TMemory::LLSTsearch
|    |    |-DVector::Getz
|    |    |-DVector::Getx
|    |    |-DVector::Gety
|    |    |-dbg_sqrt
|    |    |-TMaster::GetUanVocab
|    |    |-TVocab::get_first_word
|    |    |-TMessages::WriteMessage
|    |    |-KNMXcontrol
|    |    |    |-TMemory::Index2Ptr
|    |    |    |-TMemory::LLSTsearch
|    |    |    |-TMemory::Ptr2Index
|    |    |    |-operator-
|    |    |    |-TPathEntry::GetPos
|    |    |    |-DVector::GetLength
|    |    |    |-TPathEntry::TPathEntry
|    |    |    |-TPathEntry::SetNext
|    |    |    |-TPathEntry::SetPrev
|    |    |    |-TPathEntry::SetPos
|    |    |    |-TPathEntry::SetSpeed
|    |    |    |-TPathEntry::PutUnitVel
|    |    |    |-DVector::Norm
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|    |    |    |-TPathEntry::SetFlags
|    |    |    |-TPathEntry::GetFlags
|    |    |    |-DVector::DVector
|    |    |    |-TPathEntry::GetUnitVel
|    |    |    |-DVector::CrossProduct
|    |    |    |-isZeroEquiv
|    |    |    |-TPathEntry::GetSpeed
|    |    |    |-operator^
|    |    |    |-KNMXmaneuver
|    |    |    |    |-isZeroEquiv
|    |    |    |    |-FloatVector::Getz
|    |    |    |    |-operator-
|    |    |    |    |-DVector::DVector
|    |    |    |    \-DVector::GetLength
|    |    |    |-KNMXpoints
|    |    |    |    |-TMemory::Index2Ptr
|    |    |    |    |-TPathEntry::GetNext
|    |    |    |    |-operator-
|    |    |    |    |-TPathEntry::GetPos
|    |    |    |    |-DVector::GetLength
|    |    |    |    |-DVector::Norm
|    |    |    |    |-TPathEntry::GetSpeed
|    |    |    |    |-TMemory::Ptr2Index
|    |    |    |    |-isZeroEquiv
|    |    |    |    |-TPathEntry::TPathEntry
|    |    |    |    |-TPathEntry::PutUnitVel
|    |    |    |    |-TPathEntry::GetUnitVel
|    |    |    |    |-TPathEntry::SetPos
|    |    |    |    |-operator+
|    |    |    |    |-operator*
|    |    |    |    |-TPathEntry::SetSpeed
|    |    |    |    |-TPathEntry::SetTDepart
|    |    |    |    |-TPathEntry::GetTDepart
|    |    |    |    |-TPathEntry::SetTArrive
|    |    |    |    |-TPathEntry::SetFlags
|    |    |    |    |-TPathEntry::GetFlags
|    |    |    |    |-TPathEntry::SetNext
|    |    |    |    |-TPathEntry::SetPrev
|    |    |    |    |-DVector::Getz
|    |    |    |    |-DVector::DVector
|    |    |    |    |-DVector::Putz
|    |    |    |    |-DVector::operator*=
|    |    |    |    \-dbg_sqrt
|    |    |    |-TPathEntry::SetTDepart
|    |    |    |-TPathEntry::GetTDepart
|    |    |    |-TPathEntry::SetTArrive
|    |    |    |-DVector::GetHorizLength
|    |    |    |-DVector::operator
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Getz
|    |    |    |-DVector::Gety
|    |    |    |-operator-
|    |    |    |-DVector::operator+=
|    |    |    |-operator*
|    |    |    |-DVector::operator-=
|    |    |    |-operator/
|    |    |    |-operator+
|    |    |    |-TPathEntry::SetRadius
|    |    |    |-KNMXturnarc
|    |    |    |    |-TMemory::Index2Ptr
|    |    |    |    |-TPathEntry::GetUnitVel
|    |    |    |    |-TPathEntry::GetSpeed
|    |    |    |    |-isZeroEquiv
|    |    |    |    |-vecarc
|    |    |    |    |    |-dbg_acos
|    |    |    |    |    \-DVector::DotProduct
|    |    |    |    |         |-DVector::Getx
|    |    |    |    |         |-DVector::Gety
|    |    |    |    |         \-DVector::Getz
|    |    |    |    |-operator*
|    |    |    |    |-DVector::Norm
|    |    |    |    |-TPathEntry::PutUnitVel
|    |    |    |    |-TPathEntry::TPathEntry
|    |    |    |    |-TMemory::Ptr2Index
|    |    |    |    |-TPathEntry::SetNext
|    |    |    |    |-TPathEntry::SetPrev
|    |    |    |    |-dbg_sqrt
|    |    |    |    |-operator+
|    |    |    |    |-operator*
|    |    |    |    |-TPathEntry::SetPos
|    |    |    |    |-operator-
|    |    |    |    |-TPathEntry::GetPos
|    |    |    |    |-TPathEntry::SetTDepart
|    |    |    |    |-TPathEntry::GetTDepart
|    |    |    |    |-TPathEntry::SetTArrive
|    |    |    |    \-TPathEntry::SetSpeed
|    |    |    |-TPathEntry::GetNext
|    |    |    |-dbg_sqrt
|    |    |    |-TPathEntry::GetTArrive
|    |    |    |-DVector::Putz
|    |    |    \-TPathEntry::CreateOrientation
|    |    |         |-TMemory::Index2Ptr
|    |    |         |-DVector::DVector
|    |    |         |-TOrientation::NeedInitialFacing
|    |    |         |-operator-
|    |    |         |-TPathEntry::GetNextPtr
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|   |   |       |-TPathEntry::GetPos
|   |   |       |-DVector::Norm
|   |   |       |-isZeroEquiv
|   |   |       |-DVector::Getx
|   |   |       |-DVector::Gety
|   |   |       |-dbg_atan2
|   |   |       |-TOrientation::SetFacing
|   |   |       |   |-TMemory::Index2Ptr
|   |   |       |   |-MovePlane::SetRate
|   |   |       |   |   |-isEquiv
|   |   |       |   |   \-isZeroEquiv
|   |   |       |   \-MovePlane::SetValue
|   |   |       |       \-isZeroEquiv
|   |   |       |-dbg_asin
|   |   |       |-DVector::Getz
|   |   |       |-TPathEntry::CheckOBOrient
|   |   |       |   |-TMemory::Index2Ptr
|   |   |       |   |-DVector::Getx
|   |   |       |   |-DVector::Gety
|   |   |       |   |-DVector::Getz
|   |   |       |   |-isZeroEquiv
|   |   |       |   |-FloatVector::Getx
|   |   |       |   |-FloatVector::Gety
|   |   |       |   |-FloatVector::Getz
|   |   |       |   |-TOrientation::AddPoint
|   |   |       |   |   |-TOrientation::FindOrCreate
|   |   |       |   |   |   |-TOrientation::FindElement
|   |   |       |   |   |   |-TOrientElement::GetTime
|   |   |       |   |   |   |-TOrientElement::TOrientElement
|   |   |       |   |   |   \-TMemory::Ptr2Index
|   |   |       |   |   |-TOrientElement::DeleteAllAfter
|   |   |       |   |   |   |-TOrientElement::GetNextPtr
|   |   |       |   |   |   \-TOrientElement::~TOrientElement
|   |   |       |   |   |       |-TOrientElement::GetPrevPtr
|   |   |       |   |   |       |   \-TMemory::Index2Ptr
|   |   |       |   |   |       \-TOrientElement::GetNextPtr
|   |   |       |   |   |-TOrientation::UpdCurrPtr
|   |   |       |   |   |   |-TMemory::Ptr2Index
|   |   |       |   |   |   \-TOrientation::FindElement
|   |   |       |   |   |-MovePlane::SetRate
|   |   |       |   |   |-MovePlane::GetValue
|   |   |       |   |   |-MovePlane::SetValue
|   |   |       |   |   |-MovePlane::GetDuration
|   |   |       |   |   \-isZeroEquiv
|   |   |       |   \-TOrientation::RateChange
|   |   |       |       |-TOrientation::FindOrCreate
|   |   |       |       \-MovePlane::SetRate
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

```
|    |    |          |-DVector::VectorsToAngles
|    |    |          |     |-dbg_asin
|    |    |          |     |-DVector::Getz
|    |    |          |     \-dbg_atan2
|    |    |          |-TOrientation::GetLeftVector
|    |    |          |     |-TOrientation::FindElement
|    |    |          |     |-MovePlane::CalcValue
|    |    |          |     |-TOrientElement::GetTime
|    |    |          |     \-DVector::DVector
|    |    |          |-TOrientation::AddManeuver
|    |    |          |     \-TOrientation::AddPoint
|    |    |          |-TPathEntry::GetRadius
|    |    |          |-TPathEntry::ManeuverData
|    |    |          |-operator+
|    |    |          |-operator*
|    |    |          |-TPathEntry::GetUnitVel
|    |    |          |-vecarc
|    |    |          |-DVector::CrossProduct
|    |    |          |-TOrientation::CalcYaw
|    |    |          |-TOrientation::AddPoint
|    |    |          \-TOrientation::CalcRollRate
|    |    |                |-TOrientation::FindElement
|    |    |                |-MovePlane::CalcRate
|    |    |                \-TOrientElement::GetTime
|    |    \-TMemory::Deallocate
|    |-TPathEntry::GetTDepart
|    |-TTable::SearchInt
|    |-TMemory::Deallocate
|    \-TMessages::WriteMessage
|-TMemory::Deallocate
\-TMBRmutigon
```

FIGURE 2.2-2. Movement Code Tree. (Contd.)

2.2.4 Assumptions and Limitations

- Space and time are represented by Newtonian physics.
- Mass is not explicitly represented.
- Movement paths are represented by a series of straight line segments and arcs of circles.
- Acceleration due to gravity is 9.80852 m/sec/sec.
- There is no explicit capability for multiple PLAYERs to move in formation.
- Platforms in SWEG do not have mass, thus no dynamic movement limitations are imposed by the model.

2.2.5 Known Problems or Anomalies

None.